

Python: module cdms.grid

cdms.grid

[index](#)

CDMS Grid objects

Modules

Numeric	copy	string
PropertiedClasses	cdms.internattr	sys
cdtime	regrid	types

Classes

[cdms.cdmsobj.CdmsObj\(cdms.internattr.InternalAttributesClass\)](#)
[AbstractGrid](#)
[AbstractRectGrid](#)
[FileRectGrid](#)
[RectGrid](#)
[TransientRectGrid](#)

class **AbstractGrid**([cdms.cdmsobj.CdmsObj](#))

Method resolution order:

[AbstractGrid](#)
[cdms.cdmsobj.CdmsObj](#)
[cdms.internattr.InternalAttributesClass](#)
[PropertiedClasses.Properties.PropertiedClass](#)

Methods defined here:

[__init__](#)(self, node)

[__repr__](#) = [__str__](#)(self)

[__str__](#)(self)

[**checkAxes**](#)(self, axes)

Return 1 iff self.**getAxisList** and axes are consistent.

[**clone**](#)(self, copyData=1)

Make a copy of self.

[**flatAxes**](#)(self)

Return (flatlat, flatlon) where flatlat is a raveled NumPy array having the same length as the number of cells in the grid, similarly for flatlon.

getAxisList(self)

hasCoordType(self, coordType)
Return 1 iff self has the coordinate type.

info(self, flag=None, device=None)
Write info about slab; include dimension values and weights if applicable.

isClose(self, g)
Return 1 if g is 'close enough' to self to be considered equal.

listall(self, all=None)

reconcile(self, axes)
Return a grid that is consistent with the axes, or None.

size(self)
Return number of cells in the grid

subSlice(self, *specs, **keys)
Get a subgrid based on an argument list <specs> of slices.

writeScrip(self, cdunifFile)
Write a grid to a SCRIP file

writeToFile(self, file)
Write self to a CdmsFile file, returning CF coordinates attributes.

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

dump(self, path=None, format=1)
dump(self, path=None, format=1)
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
Match a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match tag.

matchone(self, pattern, attrname)
Return true iff the attribute with name attrname is a string which matches the compiled regular expression pattern.
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is None.

searchPattern(self, pattern, attribute, tag)

```

# Search for a pattern in a string-valued attribute. If attribute
# search all string attributes. If tag is not None, it must match
# the tag.

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern.
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not
    a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

```

```

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

```

```

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

```

```

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd.
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.

```

```
    if nowrite and nodelete is None: nodelete = 1
```

class ***AbstractRectGrid***(***AbstractGrid***)

AbstractRectGrid defines the interface for rectilinear grids:
grids which can be decomposed into 1-D latitude and longitude axes

Method resolution order:

```
AbstractRectGrid
AbstractGrid
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass
```

Methods defined here:

__init__(self, node)

classify(self)

```
# Generate a best guess at grid info for a single grid
# Return a tuple (type,nlats,isoffset) where:
#   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
#   nlats is the number of latitudes of the grid:
#     if gaussian, the gaussian nlats minus pole values
#     if equalarea, the equalarea nlats minus pole values
#   isoffset is true iff this is a BOUNDARY grid, hence the b
#     are the points wrt nlat, plus the poles.
```

classifyInFamily(self, gridlist)

```
# Generate a best guess at grid info within a family of grids
# Return a tuple (type,coverage,nlats,isoffset, basegrid, lat
#   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
#   coverage == ('global' | 'regional')
#   nlats is the number of latitudes of the FULL grid:
#     if gaussian, the gaussian nlats minus pole values
#     if equalarea, the equalarea nlats minus pole values
#     if regional, nlats for the full grid, of which this is
#     if offset, nlats for which this is the BOUNDARY grid
#   isoffset is true iff this is a BOUNDARY grid, hence the b
#     are the points wrt nlat, plus the poles.
#   basegrid is the full grid, if this is regional, or None
#   latindex is index into basegrid latitude, or None
```

flatAxes(self)

Return (flatlat, flatlon) where flatlat is a 1D NumPy array
having the same length as the number of cells in the grid, si
for flatlon.

genBounds(self)

```
# Generate default bounds
```

```

getAxis(self, naxis)
    # Get the n-th axis. naxis is 0 or 1.

getBounds(self)

getLatitude(self)

getLongitude(self)

getMask(self)

getMesh(self)
    Generate a mesh array for the meshfill graphics method.

getOrder(self)

getType(self)

getWeights(self)
    # Return normalized area weights, as latWeights, lonWeights:
    #   latWeights[i] = 0.5 * abs(sin(latBnds[i+1]) - sin(latBnds[i]))
    #   lonWeights[i] = abs(lonBnds[i+1] - lonBnds[i])/360.0
    # Assumes that both axes are represented in degrees.

listall(self, all=None)

setMask(self, mask, permanent=0)

setType(self, gridtype)

size(self)

subGrid(self, latinterval, loninterval)
    # Create a transient grid for the index (tuple) intervals.

subGridRegion(self, latRegion, lonRegion)
    # Same as subGrid, for coordinates

toCurveGrid(self, gridid=None)
    Convert to a curvilinear grid.
    'gridid' is the string identifier of the resulting curvilinear grid.

toGenericGrid(self, gridid=None)

transpose(self)
    # Return a transient grid which is the transpose of this grid.

writeScrip(self, cufile, gridTitle=None)
    Write a grid to a SCRIP file.
    cufile is a Cdunif file, NOT a CDMS file.
    gridtitle is a string identifying the grid.

```

writeToFile(self, file)

Data and other attributes defined here:

gridtypes = ['gaussian', 'uniform', 'equalarea', 'generic']

Methods inherited from [AbstractGrid](#):

__repr__ = ***__str__***(self)

__str__(self)

checkAxes(self, axes)

Return 1 iff self.***getAxisList*** and axes are consistent.

clone(self, copyData=1)

Make a copy of self.

getAxisList(self)

hasCoordType(self, coordType)

Return 1 iff self has the coordinate type.

info(self, flag=None, device=None)

Write info about slab; include dimension values and weights if

isClose(self, g)

Return 1 if g is 'close enough' to self to be considered equal.

reconcile(self, axes)

Return a grid that is consistent with the axes, or None.

subSlice(self, *specs, **keys)

Get a subgrid based on an argument list <specs> of slices.

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

dump(self, path=None, format=1)

dump(self, path=None, format=1)

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)

Match a pattern in a string-valued attribute. If attribute

search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)

Return true iff the attribute with name attrname is a string

attribute which matches the compiled regular expression pattern

if attrname is None and pattern matches at least one string

```

# attribute. Return false if the attribute is not found or is None.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute is None,
    # search all string attributes. If tag is not None, it must match the tag.

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches the tag.
    # If the predicate returns false, return an empty list.

searchone(self, pattern, atname)
    Return true iff the attribute with name atname is a string
    attribute which contains the compiled regular expression pattern.
    If atname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)
__getattr__(self, name)
__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd.
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.

```

```
    nowrite defaults to 0.  
nodelete = 1 prevents deleting this attribute.  
    nodelete defaults to 1 unless actd given.  
    if nowrite and nodelete is None: nodelete = 1
```

```
class FileRectGrid(AbstractRectGrid)
```

Method resolution order:

```
FileRectGrid  
AbstractRectGrid  
AbstractGrid  
cdms.cdmsobj.CdmsObj  
cdms.internattr.InternalAttributesClass  
PropertiedClasses.Properties.PropertiedClass
```

Methods defined here:

```
__init__(self, parent, gridname, latobj, lonobj, order, gridtype, maskobj=None, tempMask=None)  
getMask(self)  
    # Return the mask array (NOT the mask variable).  
getMaskVar(self)  
setBounds(self, latBounds, lonBounds, persistent=0)  
    # Set bounds. If persistent==1, write to file, else just shade.  
setMask(self, mask, persistent=0)  
    # Set the mask to array 'mask'. If persistent == 1, modify persistent  
    # in the file, else set as a temporary mask.
```

Methods inherited from AbstractRectGrid:

```
classify(self)  
    # Generate a best guess at grid info for a single grid  
    # Return a tuple (type,nlats,isoffset) where:  
    #     type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')  
    #     nlats is the number of latitudes of the grid:  
    #         if gaussian, the gaussian nlats minus pole values  
    #         if equalarea, the equalarea nlats minus pole values  
    #     isoffset is true iff this is a BOUNDARY grid, hence the boundary  
    #         are the points wrt nlat, plus the poles.  
  
classifyInFamily(self, gridlist)  
    # Generate a best guess at grid info within a family of grids  
    # Return a tuple (type,coverage,nlats,isoffset, basegrid, latobj)  
    #     type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')  
    #     coverage == ('global' | 'regional')  
    #     nlats is the number of latitudes of the FULL grid:  
    #         if gaussian, the gaussian nlats minus pole values
```

```

#      if equalarea, the equalarea nlats minus pole values
#      if regional, nlats for the full grid, of which this is
#      if offset, nlats for which this is the BOUNDARY grid
#      isoffset is true iff this is a BOUNDARY grid, hence the b
#          are the points wrt nlat, plus the poles.
#      basegrid is the full grid, if this is regional, or None
#      latindex is index into basegrid latitude, or None

flatAxes(self)
    Return (flatlat, flatlon) where flatlat is a 1D NumPy array
    having the same length as the number of cells in the grid, si
    for flatlon.

genBounds(self)
    # Generate default bounds

getAxis(self, naxis)
    # Get the n-th axis. naxis is 0 or 1.

getBounds(self)

getLatitude(self)

getLongitude(self)

getMesh(self)
    Generate a mesh array for the meshfill graphics method.

getOrder(self)

getType(self)

getWeights(self)
    # Return normalized area weights, as latWeights, lonWeights:
    #     latWeights[i] = 0.5 * abs(sin(latBnds[i+1]) - sin(latBnds[i]))
    #     lonWeights[i] = abs(lonBnds[i+1] - lonBnds[i])/360.0
    # Assumes that both axes are represented in degrees.

listall(self, all=None)

setType(self, gridtype)

size(self)

subGrid(self, latinterval, loninterval)
    # Create a transient grid for the index (tuple) intervals.

subGridRegion(self, latRegion, lonRegion)
    # Same as subGrid, for coordinates

toCurveGrid(self, gridid=None)

```

```
Convert to a curvilinear grid.  
'gridid' is the string identifier of the resulting curvilinea
```

toGenericGrid(self, gridid=None)

transpose(self)
 # Return a transient grid which is the transpose of this grid.

writeScrip(self, cufile, gridTitle=None)
 Write a grid to a SCRIP file.
 cufile is a Cdunif file, NOT a CDMS file.
 gridtitle is a string identifying the grid.

writeToFile(self, file)

Data and other attributes inherited from AbstractRectGrid:

gridtypes = ['gaussian', 'uniform', 'equalarea', 'generic']

Methods inherited from AbstractGrid:

__repr__ = ***__str__***(self)

__str__(self)

checkAxes(self, axes)
 Return 1 iff self.***getAxisList*** and axes are consistent.

clone(self, copyData=1)
 Make a copy of self.

getAxisList(self)

hasCoordType(self, coordType)
 Return 1 iff self has the coordinate type.

info(self, flag=None, device=None)
 Write info about slab; include dimension values and weights if

isClose(self, g)
 Return 1 if g is 'close enough' to self to be considered equal.

reconcile(self, axes)
 Return a grid that is consistent with the axes, or None.

subSlice(self, *specs, **keys)
 Get a subgrid based on an argument list <specs> of slices.

Methods inherited from cdms.cdmsobj.CdmsObj:

dump(self, path=None, format=1)

```

dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1

```

class ***RectGrid***(AbstractRectGrid)

Method resolution order:

RectGrid
AbstractRectGrid
AbstractGrid
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass

Methods defined here:

```

__init__(self, parent, rectgridNode=None)

getMask(self)

getMaskVar(self)

initDomain(self, axisdict, vardict)
    # Set pointers to related structural elements: lon, lat axes,

```

Methods inherited from AbstractRectGrid:

```

classify(self)
    # Generate a best guess at grid info for a single grid
    # Return a tuple (type, nlats, isoffset) where:
    #   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
    #   nlats is the number of latitudes of the grid:
    #       if gaussian, the gaussian nlats minus pole values
    #       if equalarea, the equalarea nlats minus pole values
    #   isoffset is true iff this is a BOUNDARY grid, hence the b
    #       are the points wrt nlat, plus the poles.

```

```

classifyInFamily(self, gridlist)
    # Generate a best guess at grid info within a family of grids
    # Return a tuple (type, coverage, nlats, isoffset, basegrid, lat)
    #   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
    #   coverage == ('global' | 'regional')
    #   nlats is the number of latitudes of the FULL grid:
    #       if gaussian, the gaussian nlats minus pole values
    #       if equalarea, the equalarea nlats minus pole values
    #       if regional, nlats for the full grid, of which this is
    #           if offset, nlats for which this is the BOUNDARY grid
    #           isoffset is true iff this is a BOUNDARY grid, hence the b
    #               are the points wrt nlat, plus the poles.
    #   basegrid is the full grid, if this is regional, or None
    #   latindex is index into basegrid latitude, or None

flatAxes(self)
    Return (flatlat, flatlon) where flatlat is a 1D NumPy array
    having the same length as the number of cells in the grid, si
    for flatlon.

genBounds(self)
    # Generate default bounds

getAxis(self, naxis)
    # Get the n-th axis. naxis is 0 or 1.

getBounds(self)

getLatitude(self)

getLongitude(self)

getMesh(self)
    Generate a mesh array for the meshfill graphics method.

getOrder(self)

getType(self)

getWeights(self)
    # Return normalized area weights, as latWeights, lonWeights:
    #   latWeights[i] = 0.5 * abs(sin(latBnds[i+1]) - sin(latBnds[i]))
    #   lonWeights[i] = abs(lonBnds[i+1] - lonBnds[i])/360.0
    # Assumes that both axes are represented in degrees.

listall(self, all=None)

setMask(self, mask, permanent=0)

setType(self, gridtype)

size(self)

```

```

subGrid(self, latinterval, loninterval)
    # Create a transient grid for the index (tuple) intervals.

subGridRegion(self, latRegion, lonRegion)
    # Same as subGrid, for coordinates

toCurveGrid(self, gridid=None)
    Convert to a curvilinear grid.
    'gridid' is the string identifier of the resulting curvilinear

toGenericGrid(self, gridid=None)

transpose(self)
    # Return a transient grid which is the transpose of this grid

writeScrip(self, cufile, gridTitle=None)
    Write a grid to a SCRIP file.
    cufile is a Cdunif file, NOT a CDMS file.
    gridtitle is a string identifying the grid.

writeToFile(self, file)

```

Data and other attributes inherited from [AbstractRectGrid](#):

gridtypes = ['gaussian', 'uniform', 'equalarea', 'generic']

Methods inherited from [AbstractGrid](#):

__repr__ = **__str__**(self)

__str__(self)

checkAxes(self, axes)

Return 1 iff self.**getAxisList** and axes are consistent.

clone(self, copyData=1)

Make a copy of self.

getAxisList(self)

hasCoordType(self, coordType)

Return 1 iff self has the coordinate type.

info(self, flag=None, device=None)

Write info about slab; include dimension values and weights if

isClose(self, g)

Return 1 if g is 'close enough' to self to be considered equal.

reconcile(self, axes)

Return a grid that is consistent with the axes, or None.

`subSlice(self, *specs, **keys)`
Get a subgrid based on an argument list <specs> of slices.

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

`dump(self, path=None, format=1)`
`dump(self, path=None, format=1)`
Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for readability.

`matchPattern(self, pattern, attribute, tag)`
Match a pattern in a string-valued attribute. If attribute
search all string attributes. If tag is not None, it must match.

`matchone(self, pattern, attrname)`
Return true iff the attribute with name attrname is a string
attribute which matches the compiled regular expression pattern.
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is not a string.

`searchPattern(self, pattern, attribute, tag)`
Search for a pattern in a string-valued attribute. If attribute
search all string attributes. If tag is not None, it must match.

`searchPredicate(self, predicate, tag)`
Apply a truth-valued predicate. Return a list containing a
if the predicate is true and either tag is None or matches tag.
If the predicate returns false, return an empty list.

`searchone(self, pattern, attrname)`
Return true iff the attribute with name attrname is a string
attribute which contains the compiled regular expression pattern.
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is not a string.

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

`is_internal_attribute(self, name)`
`is_internal_attribute(name)` is true if name is internal.

`replace_external_attributes(self, newAttributes)`
`replace_external_attributes(newAttributes)`
Replace the external attributes with dictionary newAttributes.

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

`__delattr__(self, name)`
`__getattr__(self, name)`

```

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1

```

class ***TransientRectGrid***(AbstractRectGrid)

Grids that live in memory only.

Method resolution order:

[TransientRectGrid](#)
[AbstractRectGrid](#)
[AbstractGrid](#)
[cdms.cdmsobj.CdmsObj](#)
[cdms.internattr.InternalAttributesClass](#)
[PropertiedClasses.Properties.PropertiedClass](#)

Methods defined here:

__init__(self, latobj, lonobj, order, gridtype, maskarray=None)

getMask(self)

setBounds(self, latBounds, lonBounds)

setMask(self, mask, persistent=0)

Set the mask. The persistent argument is provided for compa
with persistent versions, is ignored.

Methods inherited from AbstractRectGrid:

```

classify(self)
    # Generate a best guess at grid info for a single grid
    # Return a tuple (type,nlats,isoffset) where:
    #   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
    #   nlats is the number of latitudes of the grid:
    #     if gaussian, the gaussian nlats minus pole values
    #     if equalarea, the equalarea nlats minus pole values
    #   isoffset is true iff this is a BOUNDARY grid, hence the b
    #     are the points wrt nlat, plus the poles.

classifyInFamily(self, gridlist)
    # Generate a best guess at grid info within a family of grids
    # Return a tuple (type,coverage,nlats,isoffset, basegrid, lat
    #   type == ('gaussian' | 'equalarea' | 'uniform' | 'generic')
    #   coverage == ('global' | 'regional')
    #   nlats is the number of latitudes of the FULL grid:
    #     if gaussian, the gaussian nlats minus pole values
    #     if equalarea, the equalarea nlats minus pole values
    #     if regional, nlats for the full grid, of which this is
    #       if offset, nlats for which this is the BOUNDARY grid
    #   isoffset is true iff this is a BOUNDARY grid, hence the b
    #     are the points wrt nlat, plus the poles.
    #   basegrid is the full grid, if this is regional, or None
    #   latindex is index into basegrid latitude, or None

flatAxes(self)
    Return (flatlat, flatlon) where flatlat is a 1D NumPy array
    having the same length as the number of cells in the grid, si
    for flatlon.

genBounds(self)
    # Generate default bounds

getAxis(self, naxis)
    # Get the n-th axis. naxis is 0 or 1.

getBounds(self)

getLatitude(self)

getLongitude(self)

getMesh(self)
    Generate a mesh array for the meshfill graphics method.

getOrder(self)

getType(self)

getWeights(self)
    # Return normalized area weights, as latWeights, lonWeights:
    #   latWeights[i] = 0.5 * abs(sin(latBnds[i+1]) - sin(latBnds

```

```

#      lonWeights[i] = abs(lonBnds[i+1] - lonBnds[i])/360.0
# Assumes that both axes are represented in degrees.

listall(self, all=None)

setType(self, gridtype)

size(self)

subGrid(self, latinterval, loninterval)
    # Create a transient grid for the index (tuple) intervals.

subGridRegion(self, latRegion, lonRegion)
    # Same as subGrid, for coordinates

toCurveGrid(self, gridid=None)
    Convert to a curvilinear grid.
    'gridid' is the string identifier of the resulting curvilinea

toGenericGrid(self, gridid=None)

transpose(self)
    # Return a transient grid which is the transpose of this gric

writeScrip(self, cufile, gridTitle=None)
    Write a grid to a SCRIP file.
    cufile is a Cdunif file, NOT a CDMS file.
    gridtitle is a string identifying the grid.

writeToFile(self, file)

```

Data and other attributes inherited from [AbstractRectGrid](#):

gridtypes = ['gaussian', 'uniform', 'equalarea', 'generic']

Methods inherited from [AbstractGrid](#):

[__repr__](#) = [__str__](#)(self)

[__str__](#)(self)

checkAxes(self, axes)

Return 1 iff self.**getAxisList** and axes are consistent.

clone(self, copyData=1)

Make a copy of self.

getAxisList(self)

hasCoordType(self, coordType)

Return 1 iff self has the coordinate type.

```
info(self, flag=None, device=None)
    Write info about slab; include dimension values and weights if
isClose(self, g)
    Return 1 if g is 'close enough' to self to be considered equal.
reconcile(self, axes)
    Return a grid that is consistent with the axes, or None.
subSlice(self, *specs, **keys)
    Get a subgrid based on an argument list <specs> of slices.
```

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

```
dump(self, path=None, format=1)
    dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match.

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern.
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not
    # a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match.

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern.
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not
    a string.
```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```
is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.
```

```

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
        Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
        nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
        nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1

```

Functions

```

createGaussianGrid(nlats, xorigin=0.0, order='yx')
    createGaussianGrid(nlats, xorigin=0.0)
    Create a Gaussian grid, with shape (nlats, 2*nlats).
    'nlats' is the number of latitudes.
    'xorigin' is the origin of the longitude axis.
    'order' is either "yx" or "xy"

createGenericGrid(latArray, lonArray, latBounds=None, lonBounds=None, order='yx', mask=None)
    # Generate a generic (untyped) grid from lat, lon vectors

createGlobalMeanGrid(grid)
    # Generate a grid for calculating the global mean. The grid is a s
    # zone covering the range of the input grid

```

```

createRectGrid(lat, lon, order='yx', type='generic', mask=None)
    # Create a transient rectilinear grid

createUniformGrid(startLat, nlat, deltaLat, startLon, nlon, deltaLon, order='yx', mask=None)
    # Generate a uniform rectilinear grid

createZonalGrid(grid)
    # Generate a grid for zonal averaging. The grid has the same latitudes
    # as the input grid, and a single longitude.

defaultRegion()
    Return a specification for a default (full) region.

setClassifyGrids(mode)
    # Set grid classification mode. If on, gridtype is determined by the
    # classification method, regardless of the value of grid.getType()
    # (if any). If 'off', the value of .grid_type overrides the classification.

setRegionSpecs(grid, coordSpec, coordType, resultSpec)
    Modify a list of coordinate specifications, given a coordinate type
    a specification for that coordinate.
    'grid' is the grid object to be associated with the region.
    'coordSpec' is a coordinate specification, having one of the forms

        x
        (x, y)
        (x, y, 'co')
        (x, y, 'co', cycle)
        ':'
        None

    'coordType' is one of CoordinateTypes
    'resultSpec' is a list of 4-tuples of the form (x, y, 'co', cycle), or
    if no spec for the corresponding dimension type.

The function sets the appropriate coordinate in resultSpec,
in the canonical form (x, y, 'co', cycle). A CDMSError exception
is raised if the entry in resultSpec is not None.

Note that time coordinate types are not permitted.

writeScripGrid(path, grid, gridTitle=None)
    Write a grid to a SCRIP grid file.
    path is the path of the SCRIP file to be created.
    grid is a CDMS grid object.
    gridTitle is a string ID for the grid.

```

Data

`CoordTypeToLoc = {'lat': 1, 'lev': 2, 'lon': 0}`
`CoordinateTypes = ['lon', 'lat', 'lev', 'time']`

```
LatitudeType = 'lat'  
LongitudeType = 'lon'  
MethodNotImplemented = 'Method not yet implemented'  
TimeType = 'time'  
VerticalType = 'lev'
```